# frivol

Generated by Doxygen 1.8.1.2

Wed May 15 2013 03:22:49

# Contents

# Chapter 1

# Class Index

## 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 2

# Class Documentation

## 2.1 frivol::Array< T > Class Template Reference

Simple fixed-size array of elements of type T.

`#include <array.hpp>`

### Public Member Functions

- **BOOST_CONCEPT_ASSERT** ((boost::DefaultConstructible< T >))
- Array (Idx size)
- Idx getSize () const

    *Returns the size of the array.*
- void resize (Idx size)

- const T & operator[] (Idx index) const
- T & operator[] (Idx index)

### 2.1.1 Detailed Description

**template<typename T>class frivol::Array< T >**

Simple fixed-size array of elements of type T.

### 2.1.2 Constructor & Destructor Documentation

#### 2.1.2.1 template<typename T > **frivol::Array< T >::Array** ( Idx *size* )

Creates an array with all elements default-constructed.

**Parameters**

| | |
|---|---|
| *size* | The size of the array. |

### 2.1.3 Member Function Documentation

#### 2.1.3.1 template< typename T > const T & **frivol::Array**< T >::operator[] ( Idx *index* ) const

Returns reference to an element in the array.

**Parameters**

| | |
|---|---|
| *index* | The zero-based index of the element. |

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | if FRIVOL_ARRAY_BOUNDS_CHECKING is defined and 'index' overflows. |

#### 2.1.3.2 template< typename T > T & **frivol::Array**< T >::operator[] ( Idx *index* )

Returns reference to an element in the array.

**Parameters**

| | |
|---|---|
| *index* | The zero-based index of the element. |

**Exceptions**

| | |
|---|---|
| *std::out_of_range* | if FRIVOL_ARRAY_BOUNDS_CHECKING is defined and 'index' overflows. |

#### 2.1.3.3 template< typename T > void **frivol::Array**< T >::resize ( Idx *size* )

Resizes the array to size. If size decreases the extra elements are removed. If size increases, the new elements are default-constructed. The operation may assign the current elements to a new place, and therefore pointers to the array may be invalidated.

**Parameters**

| | |
|---|---|
| *size* | The new size. |

The documentation for this class was generated from the following files:

- /home/topi/unison/Asiakirjat/frivol/frivol/array.hpp
- /home/topi/unison/Asiakirjat/frivol/frivol/array_impl.hpp

## 2.2 frivol::DummyPriorityQueue< PriorityT > Class Template Reference

Simple implementation of PriorityQueueConcept.

`#include <priority_queue_concept.hpp>`

### Public Member Functions

- **BOOST_CONCEPT_ASSERT** ((boost::LessThanComparable< PriorityT >))
- **DummyPriorityQueue** (Idx size)
- Idx **pop** ()
- bool **empty** () const
- void **setPriority** (Idx key, PriorityT priority)
- void **setPriorityNIL** (Idx key)

### 2.2.1 Detailed Description

**template<typename PriorityT>class frivol::DummyPriorityQueue< PriorityT >**

Simple implementation of PriorityQueueConcept.

The documentation for this class was generated from the following files:

- /home/topi/unison/Asiakirjat/frivol/frivol/priority_queue_concept.hpp
- /home/topi/unison/Asiakirjat/frivol/frivol/priority_queue_concept_impl.hpp

## 2.3 frivol::DummySearchTree< ElementT > Class Template Reference

Simple implementation of SearchTreeConcept (a wrapper around std::list).

```
#include <search_tree_concept.hpp>
```

### Public Types

- typedef std::list< ElementT >
  ::iterator **Iterator**

### Public Member Functions

- template<typename FuncT >
  Iterator **search** (FuncT func)

### 2.3.1 Detailed Description

**template<typename ElementT>class frivol::DummySearchTree< ElementT >**

Simple implementation of SearchTreeConcept (a wrapper around std::list).

The documentation for this class was generated from the following file:

- /home/topi/unison/Asiakirjat/frivol/frivol/search_tree_concept.hpp

## 2.4 frivol::PriorityQueueConcept< X, PriorityT > Class Template Reference

```
#include <priority_queue_concept.hpp>
```

### Public Member Functions

- **BOOST_CONCEPT_ASSERT** ((boost::LessThanComparable< PriorityT >))
- **BOOST_CONCEPT_USAGE** (PriorityQueueConcept)

### Public Attributes

- Idx **size**
- Idx **key**
- PriorityT **priority**

### 2.4.1 Detailed Description

**template<typename X, typename PriorityT>class frivol::PriorityQueueConcept< X, PriorityT >**

Concept checking class for priority queues X with priority values of type PriorityT (or NIL). Priority queues are initialized with given size, and contain priority values for keys 0, 1, ..., size-1. Initially, all priority values are NIL. X must support the following operations:

- <construct>(Idx size) creates priority queue for keys 0, 1, ..., size-1.
- bool empty() const returns true if all keys have NIL priority.
- Idx pop() returns the key with lowest non-NIL priority and sets the priority of that key to NIL.
- void setPriority(Idx key, PriorityT priority) sets the priority value of 'key' to non-NIL value 'priority'.
- void setPriorityNIL(Idx key) sets the priority value of key 'key' to NIL. X may assume that PriorityT is ordered with <-operator. X may have undefined behavior if supplied keys are out of range or if pop() is called when empty() returns true.

The documentation for this class was generated from the following file:

- /home/topi/unison/Asiakirjat/frivol/frivol/priority_queue_concept.hpp

## 2.5 frivol::SearchTreeConcept< X, ElementT > Class Template Reference

```
#include <search_tree_concept.hpp>
```

### Public Types

- typedef X::Iterator **IteratorT**

### Public Member Functions

- **BOOST_CONCEPT_ASSERT** ((boost::CopyConstructible< ElementT >))
- **BOOST_CONCEPT_ASSERT** ((boost::BidirectionalIterator< IteratorT >))
- **BOOST_CONCEPT_USAGE** (SearchTreeConcept)

### Public Attributes

- ElementT **elem**

### 2.5.1 Detailed Description

**template<typename X, typename ElementT>class frivol::SearchTreeConcept< X, ElementT >**

Concept checking class for search trees X for elements of type ElementT. Search trees are sequence containers, the elements of which are iterated using iterator objects of type X::Iterator. The iterator must be a standard bidirectional iterator. X must support the following operations:

- <construct>() creates empty search tree.
- bool empty() const retuns true if the search tree is empty.
- Iterator begin() returns the iterator of the first element (or past-the-end if empty).

- Iterator end() returns the iterator past the last element.
- template<typename FuncT> Iterator search(FuncT func) searches the sequence using the supplied int(-Iterator)-function that for given iterator iter returns negative if the searched element is before iter, positive if it is after iter, and 0 if iter is the right element. If an element such that func returns 0 is found, it is returned, otherwise end() is returned.
- void erase(Iterator iter) removes element at iter. Other iterators should not be invalidated.
- void insert(Iterator iter, const ElementT& elem) inserts elem before iter. Does not invalidate any iterators. X may assume that ElementT is copy constructible.

The documentation for this class was generated from the following file:

- /home/topi/unison/Asiakirjat/frivol/frivol/search_tree_concept.hpp